Final Report

Enhanced NLP based on CNN/RNN/Transformer

Team 10

- B08502060: 翁佳菱

- B08502173: 巫秉融

I. Introduction

Natural Language Processing (NLP) makes computer comprehend natural human language, digital assistants, speech-to-text translation software, machine interact with human in a human-like manner have already become a part of daily life. At present, sentiment analysis, a critical NLP technique, is widely used to extract public opinion on specific topic.

In this paper, we investigate multiple deep learning models for sentiment analysis task. Three major deep learning architecture used in this project are convolutional neural network (CNN), long short-term memory recurrent neural network (LSTM-RNN) and Transformer.

This paper has four sections, including this introduction. A brief introduction of three NLP related papers is in **Section II**. **Section III** focuses on the adaptation of general deep learning approaches to sentiment analysis task. Finally, **Section IV** and **Section V** compares the overall performance of these models and provides a conclusion on the effect of using pre-word embedding with these models.

II. Related Works

In this section, we summarize three NLP related papers, which explains the fundamental concept of word vector [1], the transformer model [2] and CNN architecture for text classification [3], respectively.

2.1 Word Embedding – word2vec

Before the method of utilizing distributional similarity, the NLP researchers used one hot encoding to package each word with vectors, assigned with specific location and feed it into neural networks. Nevertheless, dealing with huge amounts of data would enormously increase

computational cost. In addition, when we try to convey some synonyms, there should exist some similarities of its output state, but one hot encoding would only differentiate words with spatial location. Thus, people propose that models would learn the meaning of text by understanding the unknown word with its adjacent context word.

Fig. 1. An illustration of the context window [4]

Illustrate as Fig. 1 [4], we can examine the sentence “The cute cat jumps over the lazy dog.” by constructing a window that can only contain three words and scanning through the whole sentence by each step. With a large amount of training data in the corpus and keep updating weight vectors, we finally build up a model that would predict between a center word and context word by using word vectors. If each word has its own vectors, the synonyms can be described as their dot product close to one, which is one of the purposes of **word2vec**.

2.2 Transformer

Even though RNN and CNN used to be considered as the first choice when dealing with NLP problems, recently, transformer model gains lots of attention due to its relatively simple complexity and several advantages. Famous models such as BERT (Bidirectional Encoder Representations from Transformers) or GPT (Generative Pre-trained Transformer) not only performs well, it also has potential to many real-world time series applications like language translation or document generation.

Unlike RNN or LSTM, the major difference is it can handle a series of data all at once, rather than only knowing pervious word’s weights. To solve the problem of losing relationship between the head and tail information in a long sentence, transformers are able to view the whole sentence and hold position information simultaneously, in addition, the mechanism of transformer model allows for more parallelization and therefore reduce training time.

2.3 CNN for Sentence Classification

In this paper, Kim proposes that to train a simple CNN model with pre-trained word vectors (Mikolov *et al*. 2013) and keeps these word vectors static while learning task-specific vectors through fine-tuning results can achieve excellent results on multiple benchmarks.

These 4 models used for comparison are stated as below, the experiment result shows that CNN-not-static and CNN-multichannel can achieve better accuracy in average performance, which suggests that fine-tuning the pre-trained vectors for each task still gives further improvements.

1.CNN-rand: The above-mentioned model where all words are initialized at random.

2.CNN-static: Utilizing pretrain word vectors from **word2vec**, the words are randomly initialized but keep static. The model can only adjust other parameters.

3.CNN-non-static: The pre-trained vectors are fine-tuned for each task.

4.CNN-multichannel: Integrate both static and non-static model. Note that each of the channel adjusts its weight and word vectors independently.

III. Proposed Method

In this section, we explain the realization of the concepts and models mentioned in Section 2.

3.1 Word2vec techniques: Skip-gram

The essence of skip-gram is using an input center word to predict the most possible word candidate around it, which reflects the accuracy when compare to training context. Skip-gram model structure is shown as Fig. 2 [5]. Through the learning process, we are able to acquire two matrices, W_{input} and W_{output}. Each row in W_{input} describe the "nature" of the word, can used for finding relation or similarity. For instance, vec["Madrid"] - vec["Spain"] + vec["France"] is close to vec["Paris"] than other word. On the other hand, W_{output} is specifically built for the content, its dimension usually refers to the longest sentence or specific number with a view to cover whole paragraph. The goal of the model can be express as the following:

$$\text{argmax } \mathrm{p}(w_{c-1}, w_{c+1}|w_c)$$

where w_c indicate both the input and output weight vector in the center word, c+1 and c-1 show position before or after the center word in the context window. In short, it shows the probability of predicting the correct context word by giving center word. Noted that we use softmax to calculate our final possibilities to get derivatives for back propagation. Then the probability transforms to:

$$\mathrm{p}(w_{context}|w_{center}) = \frac{exp(W_{output} \cdot h)}{\sum_{i=1}^{v} exp(W_{output\,(i)} \cdot h)} \quad (1)$$

Where v is the amount of unknown words, h is the hidden word vector for giving center word.

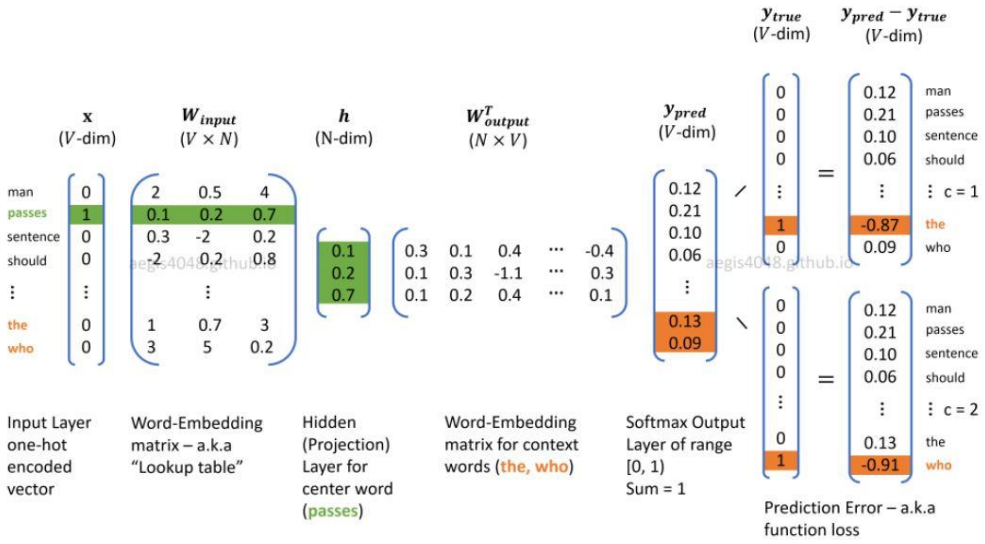


Fig. 2. Skip-gram model structure [5]

Finally, the model tries to optimize the parameter in the context window C, also taking natural log for convenient purpose

$$\text{argmax } \log \prod_{c=1}^{C} \frac{exp(W_{output} \cdot h)}{\sum_{i=1}^{v} exp(W_{output\,(i)} \cdot h)}$$

We can define our loss function as

$$J(\theta) = -\log \prod_{c=1}^{C} \frac{exp(W_{output} \cdot h)}{\sum_{i=1}^{v} exp(W_{output\,(i)} \cdot h)} = -\sum_{-c \le j \le c, j \neq 0} log\ p(w_{center+j} | w_{center}) =$$

$$-\sum_{c=1}^{C} W_{output} \cdot h + C \cdot log \sum_{i=1}^{v} exp(W_{output\,(i)} \cdot h) \quad (2)$$

$$\theta = [W_{input}\ W_{output}] = \begin{bmatrix} u_{to} \\ u_{be} \\ \vdots \\ v_{to} \end{bmatrix} \in R^{2NV} \quad (3)$$

Where θ is defined as concatenation of W_{input} and W_{output}, u and v represent weight in W_{input} and W_{output}. Normally, window size C is ranging from 1 to 10, and basic assumption of window size is the further it apart from center word, the smaller information it will carry.

Example and Explanation of Skip-gram structure

In general, the method applies forward and backward propagation to adjust the value in both W_{input} and W_{output}. Once we get our predicted value and error (y_{pred}-y_{true}), we use stochastic gradient descent (SGD) to update by running BP with small batch of sample. (Usually, NLP training data contains millions of words, considering computational cost and iteration time, it's not suitable for batch gradient descent.)

Forward propagation

Let's look at the sentence *"To be or not to be, that is a question."* Compare it with Fig. 2, the sentence has eight unique words (V=8) and we set the size of window as N=3. That way, we get an

initial W_{input} with dimension of 8x3.

To train the model, we feed it with one-hot encoded vector x of Vx1 dimension that only fill center word with 1 and others are 0. Noted that each row of W_{input} shows the values of that dimension. Usually, the dimension would up to around 200~600. In order to help the model converge faster, we can load the initial row values in W_{input} by matching it with pretrained vectors from sources like **word2vec** or **fastText**. After that, we multiply W_{input} with x so we can get projected layer h

$$h = W_{input}^{T} \cdot x \tag{4}$$

Follow the previous method, we can further multiply W_{output} with h, after treating the value with softmax, the summation of y_{pred} turn to 1. (See eq 1) The final prediction is obtained by selecting the largest value among all.

Noted that the last step has a problem that V is too large to calculate and store through softmax, we usually implement with another method called Negative Sampling to reduce computational cost.

Backward propagation

The backward propagation is based on SGD method, which updated the weights only using small batch of data, that is

$$\theta_{new} = \theta_{old} - \eta \, \nabla_{J\,(\theta, w_t)} \tag{5}$$

Where J is the loss function as mentioned before. Then, each of the weight vector is adjust by

$$W_{input}^{new} = W_{input}^{old} + \eta \frac{\partial J}{\partial W_{input}} = W_{input}^{old} - \eta x \sum_{c=1}^{C} e_c W_{output} \tag{6}$$

$$W_{output}^{new} = W_{output}^{old} + \eta \frac{\partial J}{\partial W_{output}} = W_{output}^{old} - \eta \, h \sum_{c=1}^{C} e_c \tag{7}$$

Where

$$\frac{\partial J}{\partial W_{input}} = \frac{-\partial \sum_{c=1}^{C} e_c W_{output} \cdot W_{input}^{T} \cdot x}{\partial W_{input}} = -x \sum_{c=1}^{C} e_c W_{output} \tag{8}$$

$$\frac{\partial J}{\partial W_{output}} = \frac{-\partial \sum_{c=1}^{C} W_{output} \cdot h \cdot e_c}{\partial W_{output}} = h \sum_{c=1}^{C} e_c \tag{9}$$

e_c is the error rate (y_{pred}- y_{true}) for context word in the context window.

3.2 Transformer - Model Architecture and Self Attention

The architecture of transformer is shown as Fig. 3 [2], transformer is constructed based on several layers of encoder and decoder. It has a unique mechanism called self-attention, which means

the attention layer can access all the other input data. After finishing word embedding, a series of vectors are sent into Multi-Head Attention layer in Fig. 4 [6]

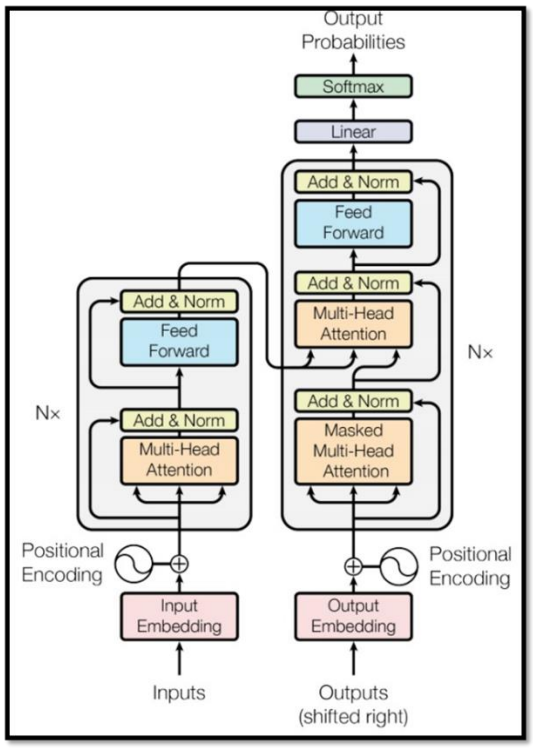


Fig. 3 Transformer architecture [2]

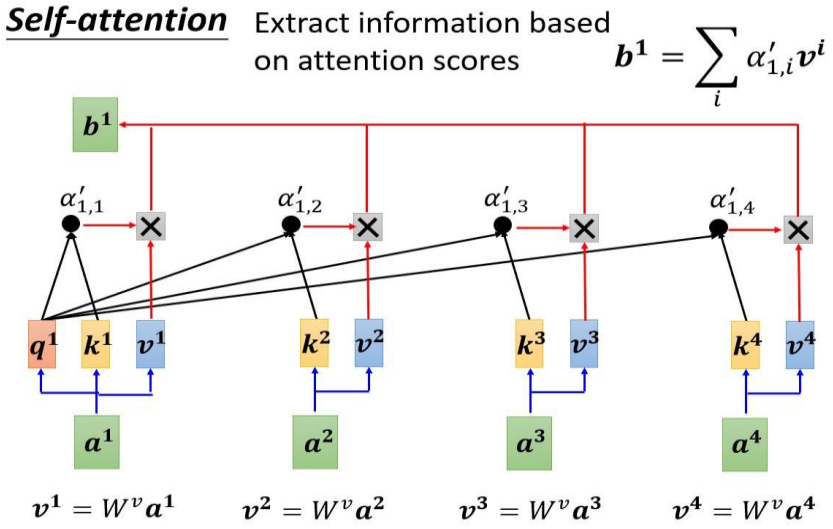


Fig. 4. Self-attention layer [6]

As shown in Fig. 4, each input a^n matches with three types of weights: key(k), query(q), and value(v). During every iteration, query q^n will perform dot product with other corresponding key value k^n, the concept can be portrayed as a word finding its associated context word. After finding the relation between a word and other context, to prevent some of the product grows too large and lead to extremely small gradients, the value will divide by $\frac{1}{\sqrt{d_k}}$, where d_k is the dimension of k

matrix (or input data). Finally, sent to softmax function to let the summation of α equals to one. Multiply α with the original value v, feed them to a fully connected neural network, and summed it to get the final attention scores, through decoded, it will generate a single output representation

Now since that the procedure is constantly doing same process with identical weight value, we can combine all the q, k, v in into matrix, which substantially increase the computation speed when using GPU. An illustration is shown in Fig. 5. In addition, "Multi-Head" means there are several dimensions of matrices processed parallelly and independently at the same time, which experimentally show a stable outcome.

Noted that the mechanism of dot product is like the search of finding most related word (also can be descried as cosine similarity), the bigger the dot product is, the more likely the outcome would be. In the "Attention is all you need" paper, the authors add position embedding to make sure each input contain position information before further processing. The position formula is show as the following.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{10}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{11}$$

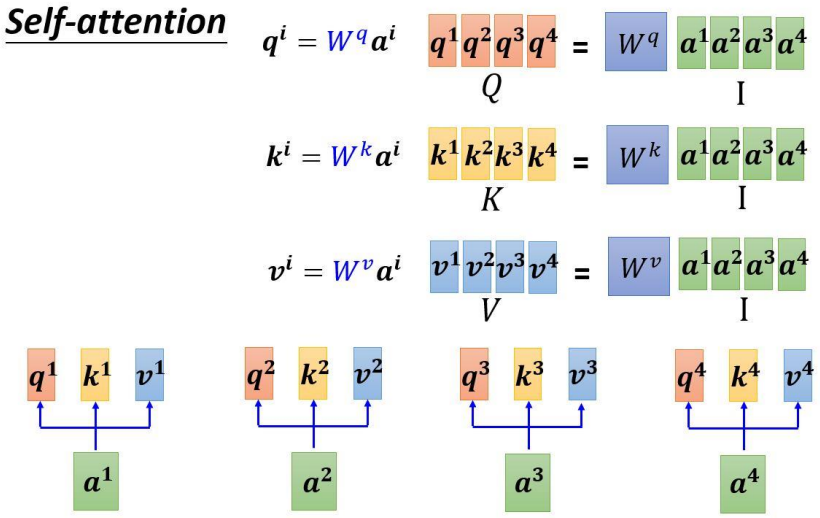


Fig. 5. Matrix computation [6]

The major difference between encoder and decoder part is masked Multi-head attention layer. In contrast to time sequence processing adopted by recurrent neural network, the layer conducts parallel processing, but input word is replaced with previous predicted output and position information. Since we can get the expected output by adjusting the process, the potential of attending subsequent position thus is avoided, like demonstration of fig.6. If the input has a sequence of A,B,C, and D, we can hide the information by resigning the later on information to be minus infinity, that way, after softmax function, matrix indices in $M_{ij,i>i}$ will turn to zero. In other word, we only use prediction A to predict B, [A, B] to predict C, [A, B, C] to predict D.

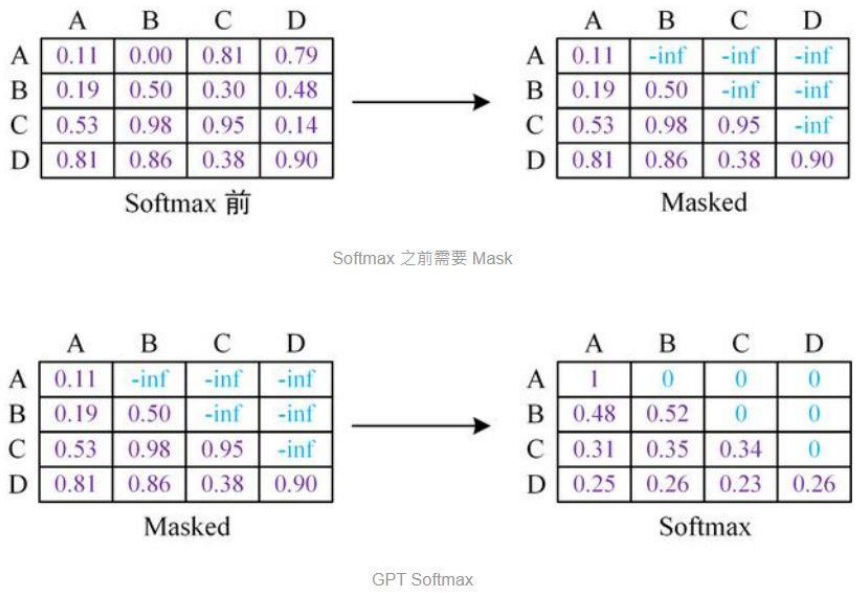


Fig. 6. Masked mechanism [7]

BERT (Bidirectional Encoder Representations from Transformers)

In the recent years, BERT (Bidirectional Encoder Representations from Transformers), which is proposed by Google, has gain popularity and performs with outstanding performance in GLUE (The General Language Understanding Evaluation) test.

BERT itself is based on a stack of encoders. (for $BERT_{base}$ has 12 layers, and $BERT_{large}$ has 24 layers) The reason that makes it unique is it tries to construct a language model that would quickly adapt to various language-oriented tasks that only need fine tuning. Instead of building all kinds of models for each NLP task, BERT can save lots of time and computation when training a model from scratch, and the parameters obtained from pre-train model makes it easier to converge.

There are two tasks for BERT during pre-train phase: masking input and next sentence prediction. During the pre-train stage, some words in the input sentence will be concealed randomly, and feed into a series of encoder, calculating dot product with self-attention mechanism, then go through linear transform (matrix), gain the final possibility of candidate words after Softmax. Since we already know the answer (just block it intentionally), then can compare errors and update the parameters in linear and encoder. Another task for BERT is trying to let the model realize the relation between sentence. There will be [CLS] and [SEP] token existed each data, [CLS] can be imagined as a classification category we are going to predict, [SEP] is just for separating the sentence. Once we get the attention score from all words in each sentence, the model will decide whether it shows connection between each sentence by same procedure.

It takes many days to get pre-train vectors due to the enormous parameters it need to optimize ($BERT_{base}$ has 110M , and $BERT_{large}$ has 340M) , nevertheless, we can utilize the open source model to fine tune specifically for our task. In supervised sentiment analysis shown in Fig. 7, the [CLS] token will be response for producing output classification (“pos” and “neg”) after doing a series of self-attention.

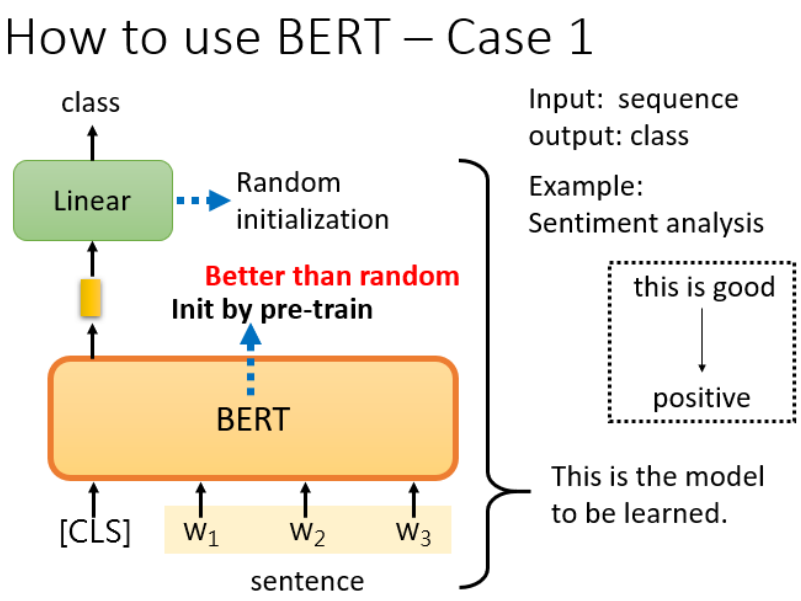


Fig.7 Fine Tuning BERT with sentiment analysis [6]

3.3 CNN for Sentence Classification – Architecture

The architecture of this CNN model is shown as Fig. 7 [3]. For a sentence of length n, x_i represent word vector corresponding to the i-th word in the sentence after tokenizing (padding if needed). Concatenate the words vector to become a matrix where horizontal axis k means word vector, and the vertical axis will have preset maximum length n. The convolution operation is done by multiple filters with different vertical length h. Considering weight w and bias b, the feature c_i is produced afterwards.

$$c_i = f(w \cdot x_{i:i+h-1} + b) \tag{12}$$

Where f is a non-linear function.

The pooling process extracts maximum value from concatenated feature map and passes to a fully connected layer with regularization. In the penultimate layer $z = \{\widehat{c_1}, \widehat{c_2} \ldots \widehat{c_m}\}$, the author employ dropout by multiplying Bernoulli random variables with probability p, that

$$y = w(z \cdot r) + b \tag{13}$$

Note that the backpropagation will not be affected by dropout, and the learned weight will be constrained with l2 norm whenever ||w|| > s.

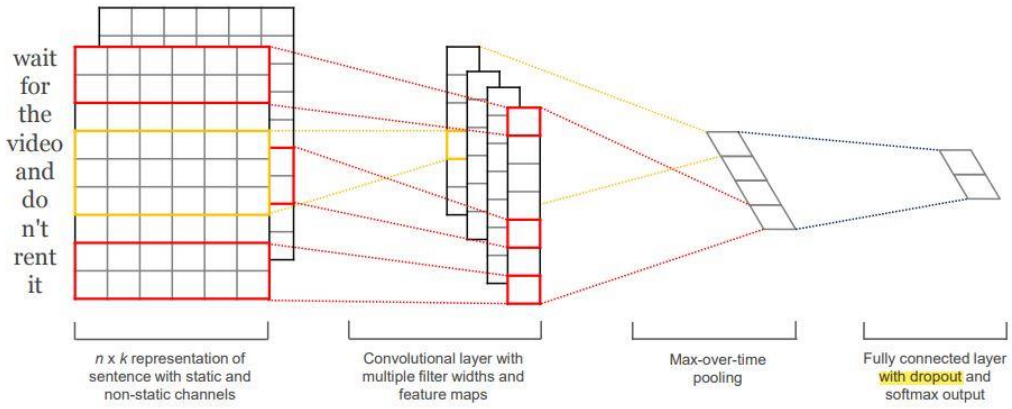


Fig. 7. CNN architecture for sentence classification [3]

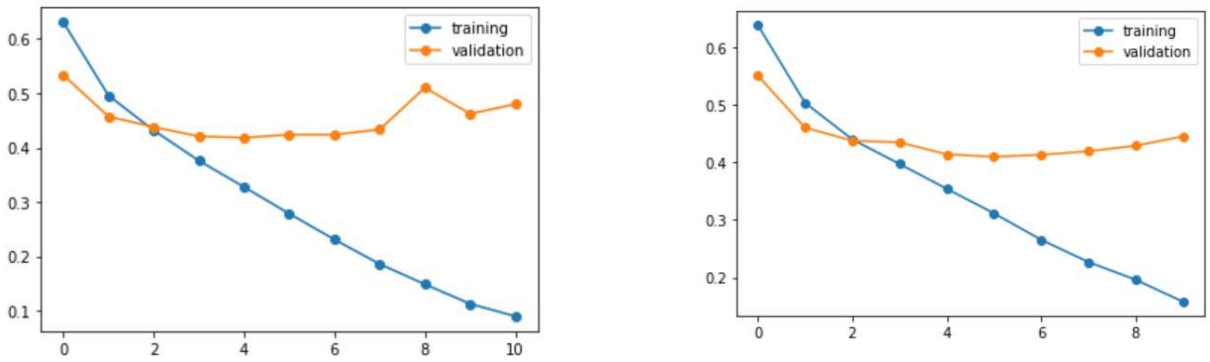
IV. Experimental Results

The source code is based on Kim's research in 2014 [3]. The dataset we use is Movie Review (MR, Pang and Lee, 2005) and the pre-trained vectors are trained by Mikolov et al. (2013) on 100 billion words of Google News .

Shown as Table. 1, original setting of the filter sizes is (3,4,5). To investigate the effect of filter size on the analysis result, we repeat the experiment with filter sizes is (2,3,4). We can observe that CNN-non-static have the best performance in these three experiments, and CNN-rand the worst. Model training loss is shown as Fig.8, which shows that change of filter sizes indeed affects the result.

Filter size	(3,4,5)	(3,4,5)	(2,3,4)
Model	**Kim**	**Our results**	**Our results**
CNN-rand	76.1	74.1	75.2
CNN-static	81.0	81.5	81.9
CNN-non-static	81.5	82.3	83.1

Table. 1. Sentiment analysis comparison with the original research paper



Filter size = (3,4,5) Filter size = (2,3,4)

Fig. 8. CNN-non-static model training loss with different filter sizes

In addition, we build RNN-LSTM model and transformer model for the same task. The result is shown as Table. 2. We can observe the same trend that non-static model performs the best and random embedding model the worst. In addition, the overall performance of the transformer model is better than the CNN and RNN model.

RNN-rand	RNN-static	RNN-non-static
73.2	77.1	78.3
BERT-rand	BERT -static	BERT -non-static
76.7	85.2	88.1

Table. 2. Sentiment analysis comparison of RNN and BERT model

V. Conclusion

From the results shown in Section 4, we can conclude that pre-trained word embedding noticeably increases the accuracy of the models on sentiment analysis, and fine-tuning can further improve the result.

For the comparison of the three models, transformer have the best result, and RNN the worst. The advantage of transformer is that it solves the long-range context dependency problem, sentences are processed at once rather than word by word while keeping the position information. As for CNN and RNN models, the reason that CNN outperformers RNN is that it efficiently extracts the most important features, and it can achieve excellent performance using only one layer.

To further improve the model, one simple way is to add the number of layers of the CNN and RNN model, in this experiment, we build CNN with 1 layer and RNN-LSTM 2 layers. Since adding number of layers of the models will increase both performance and the complexity of the models, we try to seek for improvement in the word embedding step.

The word embedding method used in the reference paper is word2vec, and here we try using fastText instead, the main difference of these two methods is that fastText treat each word as composed of character grams (sub-string), its word vector is the sum of these sub-string information. Therefore, using fastText method may find corresponding word vector even though the word does not exist in the training dataset, while using word2vec cannot. Having more pre-trained word vectors representation hold more information, thus improve the result, Table.3 is the result of using fastText pretrained vectors instead of word2vec. We can observe that the models perform better with fastText pretrained vectors, which matches our assumption.

Filter size	(3,4,5)	(3,4,5)	(2,3,4)	(3,4,5)	(2,3,4)
Model	**Kim**	**word2vec**	**word2vec**	**fastText**	**fastText**
CNN-rand	76.1	74.1	75.2	74.5	75.3
CNN-static	81.0	81.5	81.9	82.7	83.6
CNN-non-static	81.5	82.3	83.1	83.3	83.7

Table. 2. Comparison using word2vec and fastText method

VI. Task proportion

Student ID	Contribution (%)
B08502060	50%
B08502173	50%

VII. Demo

Video: https://youtu.be/gWTHmEyE6Pk

Code: https://colab.research.google.com/drive/11_sbBCX8mG9U_GKuOygX_8kW6qYAZ1Qi?usp=sharing

Reference

Research paper:

[1] Tomás Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean:Efficient Estimation of Word Representations in Vector Space. ICLR (Workshop Poster) 2013

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000–6010.

[3] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Other references:

[4] Word Embeddings. (n.d.). https://cbail.github.io/textasdata/word2vec/rmarkdown/word2vec.html

[5] Demystifying Neural Network in Skip-Gram Language Modeling. (n.d.). https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

[6] Lee, H.Y. (2022, Spring) *Self-Attention.* ML 2022 Spring self-attention lecture.

[7] A Look into OpenAI GPT and GPT2 (2021). https://baijiahao.baidu.com/s?id=1652093322137148754&wfr=spider&for=pc

Source code

https://chriskhanhtran.github.io/posts/cnn-sentence-classification/